RAIRO-Oper. Res. 42 (2008) 215–228
DOI: 10.1051/ro:2008010

RAIRO Operations Research
www.rairo-ro.org

NEW REPRESENTATION TO REDUCE THE SEARCH SPACE FOR THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM *

Khaled Moumene[1] and Jacques A. Ferland[1]

Abstract. This paper describes a new representation for the solutions of the resource-constrained project scheduling problem (RCPSP) denoted Activity Set List. The most efficient heuristics for the problem use the activity list representation and the serial SGS method to construct the corresponding solution (schedule). The activity list may induce a search space of representations much larger then the space of schedules because the same schedule can correspond to many different activity list representations. We indicate how the activity set list representation can significantly reduce the search space, and how to move more efficiently through it. Furthermore, this new representation never excludes the optimal solution and it has many interesting properties. An evaluation of the search space reduction induced by this representation is made for the most used library of instances in the literature. The activity set list representation may be used to construct a new category of more efficient solution procedures for the problem.

Keywords. Project scheduling, Resource-constrained project scheduling, Activity list representation, Activity set list representation, Heuristics and metaheuristics.

Mathematics Subject Classification. 90B35.

Received June 01, 2005. Accepted November 28, 2007.

* *This research was supported by NSERC grant (OGP 0008312).*

[1] Dept. Informatique et Recherche Opérationnelle, Université de Montréal, C. P. 6128, Succursale Centre-Ville, Montréal (Québec), H3C 3J7, Canada; khaled_moumene_75@yahoo.com; ferland@iro.umontreal.ca

Article published by EDP Sciences

1. Introduction

In a Resource-Constrained Project Scheduling Problem (RCPSP), we consider a project including a set $I = \{1, 2, \cdots, N\}$ of N activities to be scheduled under resource and precedence constraints. The time required to complete an activity i is specified in terms of an integer number d_i of periods. Note that 1 and N are used as the starting and ending activities of the project, respectively, and their duration $d_1 = d_N = 0$. It is also assumed that once initiated, any activity is completed without interruption.

On the one hand, the precedence constraints are induced by technological requirements to impose that any activity i must be scheduled for execution after the completion of all its immediate predecessors included in a set P_i. On the other hand, each activity i requires r_{ik} units of resource $k \in R = \{1, 2, \cdots, K\}$ during each period of its execution. The resource constraints are specified to limit the number of units of each resource $k \in R$ used to the number A_{kt} of units available in each period t.

The (RCPSP) is to determine a starting period for each activity in order to minimize the total duration (makespan) of the project while satisfying the precedence and resource constraints. Note that this problem is denoted as $m/cpm/c_{max}$ or $PS/prec/C_{max}$ referring to Herroelen *et al.* [14] and Brucker *et al.* [7] classifications, respectively.

Since this problem is known to be a NP-Hard [3], solving problems having more than 60 activities and 4 different resources with an exact method is very much time consuming. Several authors have proposed different solution procedures for the (RCPSP). They can be classified into three categories. The exact methods [9,12,23,24,27,28] are, essentially, implementation of the branch-and-bound to solve small problems having less than 60 activities. Using heuristic procedures is an alternative to deal with larger problems [4,9,11,17,19,20,29]. These methods are serial or parallel schedule generation schemes. The third category includes metaheuristics like tabu search [2,25], simulated annealing [5,6,8] and genetic algorithm [1,13,16].

In this paper we introduce a new representation for the solutions of (RCPSP) called *activity set list*. The most efficient heuristics for the (RCPSP) are using activity list representation [18,21] and the serial SGS method [21] to construct the corresponding solution (schedule). The activity list may induce a search space of representations much larger then the space of schedules because the same schedule can correspond to many different activity list representations. We show how the activity set list representation can considerably reduce the search space and how to move more efficiently through it. We also show that this new representation never excludes the optimal solution and that it has many interesting properties.

Computational experiments show the efficiency of the activity set list. An evaluation of the search space reduction induced by this representation is made for the most used library of instances in the literature. An analysis of this reduction is also made according to their complexity level. The activity set list representation

may be used to construct a new category of more efficient solution procedures for the problem.

The paper is organized as follows. In Section 2, we briefly describe the activity list representation and then define the activity set list representation. We introduce a powerful property of the new representation in Section 3 together with some results and procedures related to it. We show in Section 4 how the new representation can be extended for the backward scheduling mode. Finally, a performance evaluation is made in Section 5 to show the importance of the search space reduction. The numerical results for all instances in J30, J60, J90 and J120 of the PSPLIB [22] indicate the gain in efficiency when the new representation is used.

2. Activity Set List Representation

An essential feature of any heuristic solution procedure is the *encoding* or the *representation* of the solution. Note that many different procedures may use the same representation, but they differ in the way the solution representation are modified at each iteration. Several representations have been proposed [21], but one of the most commonly used is the *activity list representation* or *permutation based representation* [18,21] where we use the following vector :

$$AL = [i_1, i_2, \cdots, i_N]$$

corresponding to a permutation of the activities satisfying the precedence constraints. Hence any activity i_j appears in AL after all its immediate predecessors P_{i_j}; *i.e.*,

$$P_{i_j} \subset \{i_1, i_2, \cdots, i_{j-1}\}$$

The activity list AL can be decoded into a unique schedule using the serial SGS method [21] as follows : activity i_1 is scheduled to start in the first period (*i.e.*, at time 0). Activity i_j is the j^{th} activity scheduled as early as possible after the completion of all its predecessors in P_{i_j} according to the resource availability.

To illustrate the activity list representation, consider the example in Figure 1 where the project includes 10 activities requiring a unique type of resource. Assume that $A_{1t} = 6$ units of resource are available in each period t. With each activity-node i, we associate the pair (d_i, r_{i1}). Using the serial SGS method, the activity list,

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

is decoded into the unique feasible schedule illustrated in Figure 2. It is easy to verify that this schedule corresponds also to the other activity list,

$$[1, 2, 5, 3, 4, 6, 8, 7, 9, 10].$$

This example illustrates the fact that the heuristic procedures working with this representation may generate several different activity lists corresponding to the same feasible schedule. Thus, these procedures search in a much larger space of activity lists than the space of feasible schedules. Now the following *activity set list representation* has the advantage of reducing the search space by representing

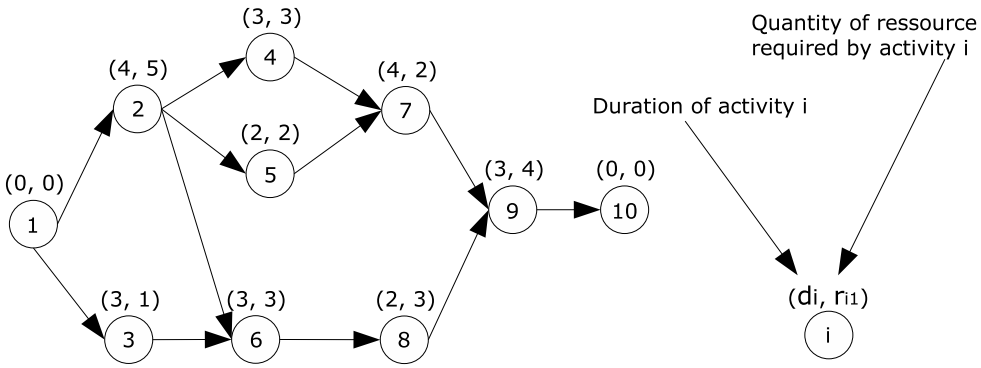

FIGURE 1. Example of (RCPSP).

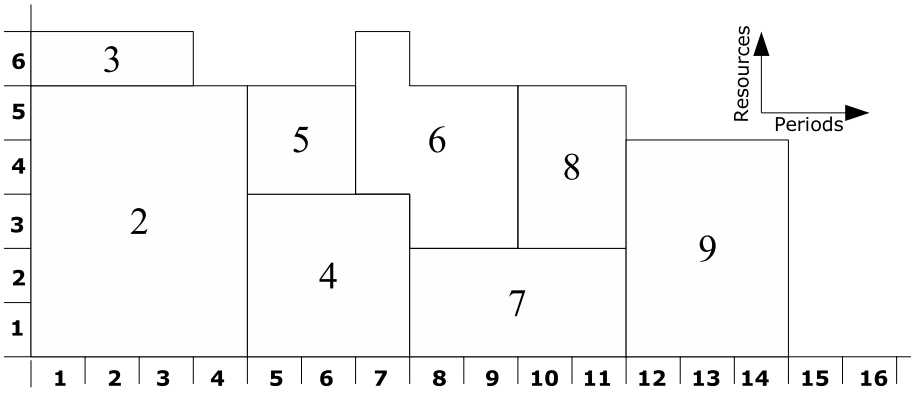

FIGURE 2. Example of a schedule.

several activity lists into a unique activity set list. To specify the activity set list representation we refer to the following definitions.

Definition 1. An activity set list ASL is an ordered list of different non empty subsets of activities $SA_1, SA_2, \cdots, SA_p$

$$\text{ASL} = [SA_1, SA_2, \cdots, SA_p]$$

such that

- Each activity $i \in I$ belongs to a unique subset SA_α, $\alpha \in \{1, 2, \cdots, p\}$ (*i.e.*, the set of subsets $SA_1, SA_2, \cdots, SA_p$ is a partition of I);
- If activity $i \in SA_\alpha$, then each of its predecessors is either in SA_1 or SA_2 or $\cdots$ or SA_α (*i.e.*, $P_i \subset SA_1 \cup SA_2 \cup \cdots \cup SA_\alpha$).

It is easy to verify that the activity list AL is an activity set list ASL where each subset SA_j, $j = 1, 2, \cdots, N$, includes a *unique* activity.

TABLE 1. Ordered lists generated from SA_1 and SA_2.

Subsets	Sets of ordered lists
$SA_1 = \{1, 2, 3, 4, 5\}$	$OS_1 = \{[1, 2, 3, 4, 5], [1, 2, 3, 5, 4], [1, 3, 2, 4, 5], [1, 3, 2, 5, 4], [1, 2, 4, 5, 3], [1, 2, 5, 4, 3], [1, 2, 4, 3, 5], [1, 2, 5, 3, 4]\}$
$SA_2 = \{6, 7, 8, 9, 10\}$	$OS_2 = \{[7, 6, 8, 9, 10], [6, 7, 8, 9, 10], [6, 8, 7, 9, 10]\}$

Definition 2. Consider a subset SA_α, an element of an activity set list ASL.

- os_α denotes an ordered list of activities of SA_α satisfying the precedence constraints (*i.e.*, each activity $i \in SA_\alpha$ appears in os_α after its predecessors in SA_α $(P_i \cap SA_\alpha)$).
- OS_α is defined as the set of all ordered lists os_α that can be generated from SA_α.

Definition 3. An activity set list ASL$= [SA_1, SA_2, \cdots, SA_p]$ has the permutation property (or ASL is (PRMT)) if all vectors $[os_1, os_2, \cdots, os_p] \in OS_1 \times OS_2 \times \cdots \times OS_p$ correspond to the same unique schedule.

To illustrate the definitions, consider the example in Figure 1. The activity set list

$$[\{1, 2, 3, 4, 5\}, \{6, 7, 8, 9, 10\}]$$

includes the subsets $SA_1 = \{1, 2, 3, 4, 5\}$ and $SA_2 = \{6, 7, 8, 9, 10\}$. Now, consider the ordered lists $os_1 = [1, 2, 5, 3, 4]$ and $os_2 = [6, 8, 7, 9, 10]$ satisfying the precedence constraints. Then the vector $[os_1, os_2] = [1, 2, 5, 3, 4, 6, 8, 7, 9, 10]$ is an activity list corresponding to the schedule of Figure 2. It is easy to verify that the combination of any ordered lists $[os_1, os_2] \in OS_1 \times OS_2$ in Table 1 is an activity list corresponding to the same unique schedule of Figure 2 (there are $8 \times 3 = 24$ possible combinations). Hence, the preceding activity set list is (PRMT). This example illustrates the advantage of working with activity set lists being (PRMT) to reduce the search space of a procedure.

Note that Definition 1 includes two conditions for a representation to be an activity set list. For instance, referring to the Figure 1, the following representation

$$[\{1, 2, 3, 7, 8\}, \{4, 5, 6, 9, 10\}]$$

is not an activity set list since the predecessors of 7 (4 and 5) belong to the second subset.

Moreover, all vectors induced by an activity set list can be activity lists but they may not correspond to the same unique schedule. Such an activity set list is not (PRMT). For instance, referring to Figure 1, the following activity set list does not verify the Definition 3 condition.

$$[\{1, 2, 3, 4, 5, 6\}\{7, 8, 9, 10\}]$$

We can easily verify that all vectors $[os_1, os_2] \in OS_1 \times OS_2$ are activity lists. Now, consider the activity lists

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

- Let AL=$[i_1, i_2, \cdots, i_N]$ the activity list to initialize the procedure.
- For any $j = 1, 2, \cdots, N$, considering the starting period of the previous activities $i_1, i_2, \cdots, i_{j-1}$ in AL, denote :
  - $TP(i_j)$: The earliest starting period of i_j accounting only for the precedence constraints.
  - $TPR(i_j)$: The earliest starting period of i_j accounting for both the precedence and the resource constraints.

1. Initialize the first subset $SA_1 = \{i_1\}$.
2. $l = 1$.
3. **For** $k = 2$ **to** N **do***
   - **if** $(TP(i_k) = TPR(i_k))$ **then** $SA_l = SA_l \cup \{i_k\}$
   - **else**
     - $l = l + 1$
     - Initialize a new subset $SA_l = \{i_k\}$

(1) The activity set list generated is $[SA_1, SA_2, \cdots, SA_l]$

FIGURE 3. The ConstructASLPRMT procedure.

$$[1, 2, 3, 4, 6, 5, 7, 8, 9, 10]$$

from $OS_1 \times OS_2$. They correspond to different schedules that have makespan of 14 and 16, respectively.

3. Generating Activity Set List being (PRMT)

Next, we propose a procedure to generate activity set list being (PRMT). Starting with an activity list AL, select each activity sequentially. Initialize the first subset of the partition with the first activity in AL. Considering the starting period of the previous activities in AL, if the earliest starting period of the activity accounting only for the precedence constraints is the same if we account for both precedence and resource constraints, then the activity is included in the current subset. Otherwise, a new subset is initialized with this activity. The procedure is summarized in Figure 3.

Note that the loop * is completed in $O(N)$. Furthermore, the values $TP(i_j)$ and $TPR(i_j)$ are evaluated in $O(cN^2)$, where c is a positive real number. Thus, ConstructASLPRMT is completed in polynomial time.

The procedure can induce less computational effort if AL has to be decoded. Indeed to generate the corresponding schedule, the activities in AL have to be selected sequentially and the values $TP(i_j)$ and $TPR(i_j)$ have to be evaluated. Hence, testing the if $TP(i_j)$ is equal to $TPR(i_j)$ for each selected activity i_j to determine its subset, is the only additional computation effort.

Referring to the example of Figure 1, it is easy to verify that starting with the activity list

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

the procedure ConstructASLPRMT generate the following activity set list being (PRMT)

$$[\{1, 2, 3, 4, 5\}, \{6, 7, 8, 9, 10\}]$$

It is easy to show that ConstructASLPRMT *generates an activity set list being (PRMT)* when initiated with an activity list. The basic argument relies on the fact that once the first activity of the subset is scheduled, then the rest of the activities in it are scheduled at the earliest period as if there were no resources constraints. As a consequence, it follows that *if an activity set list generated with ConstructASLPRMT has a unique subset, then the optimal schedule can be obtained with the CPM method.* Finally, since Kolisch [18], Sprecher *et al.* [26] have proved that if the (RCPSP) has an optimal schedule, there always exists an activity list corresponding to an optimal schedule when using the serial SGS method, than it follows that if the ConstructASLPRMT procedure is initiated with this activity list, then it generates an activity set list (PRMT) corresponding to the same optimal schedule.

4. Activity Set List Representation for the backward scheduling mode

So far, we have used the (forward) serial SGS introduced in Section 2 to decode an activity list into a schedule. Accordingly, we have introduced the notion of activity set list representation in reference with this scheduling mode. But an activity list can also be decoded into another schedule using a backward scheduling mode. This is also a serial SGS method where the last activity i_N is first scheduled. Activity i_{N-j+1} is the j^{th} activity scheduled as late as possible to be completed before starting its successors according to the resource availability.

Referring to the example of the Figure 1, the backward scheduling mode decodes the activity list

$$[1, 2, 5, 3, 6, 8, 4, 7, 9, 10]$$

into the schedule illustrated in Figure 4 which is different from the schedule, obtained with the forward mode, in Figure 5.

The notion of activity set list representation being (PRMT) can be extended in reference with the backward scheduling mode. To specify this new representation, we refer to the definitions 1 and 2 in Section 2, but we modify Definition 3 as follows :

Definition 4. An Activity Set List $ASL = [SA_1, SA_2, \cdots, SA_p]$ has the backward permutation property (or ASL is (PRMTb)) if all vectors $[os_1, os_2, \cdots, os_p] \in OS_1 \times OS_2 \times \cdots \times OS_p$ correspond to the same unique schedule when the backward scheduling mode is used.

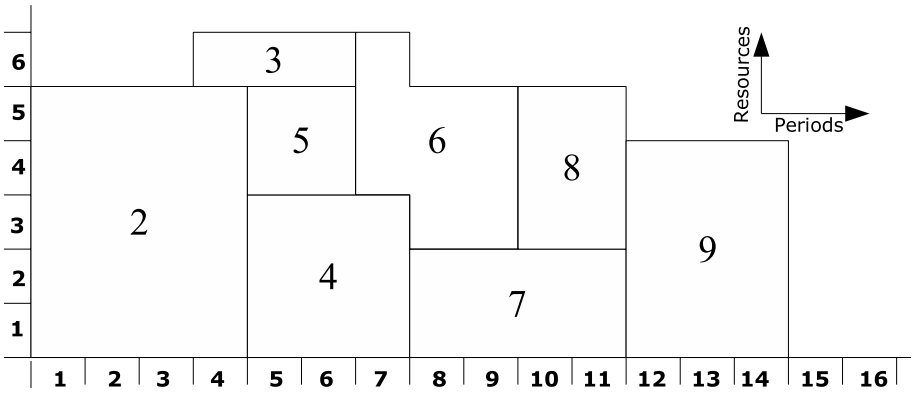

FIGURE 4. Schedule with the backward mode.

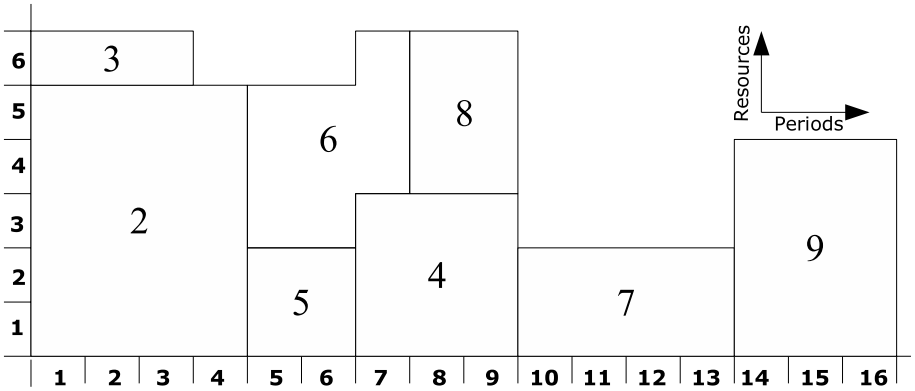

FIGURE 5. Schedule with the forward mode.

Similarly, the ConstructASLPRMT procedure can be adapted into the ConstructASLPRMTb procedure to generate an ASL being (PRMTb) starting with an activity list $AL = [i_1, i_2, \cdots, i_N]$. It can be summarized as follows. Initialize the first subset SA_1 of the partition with i_N. Considering the starting period of the following activities $i_{j+1}, i_{j+2}, \cdots, i_N$, if the latest starting period $LP(i_j)$ of activity i_j accounting only for the precedence constraints is the same as $LPR(i_j)$ if we account for both precedence and resource constraints, i_j is included in the current subset. Otherwise, if $LPR(i_j) < LP(i_j)$, a new subset is initialized with i_j. A similar result as for ConstructASLPRMT holds. Any activity set list, generated by ConstructASLPRMTb,

$$[SA_l, SA_{l-1}, \cdots, SA_1]$$

is (PRMTb).

If we initialize the ConstructASLPRMTb procedure with the activity list $[1, 2, 5, 3, 6, 8, 4, 7, 9, 10]$, it is easy to verify that the activity set list generated is

$$[\{1, 2, 5\}\{3, 6, 8, 4, 7, 9, 10\}]$$

Note that this activity set list (PRMTb) is different from the activity set list (PRMT)

$$[\{1, 2, 5, 3, 6, 8\}\{4, 7, 9, 10\}]$$

generated with ConstructASLPRMT initialized with the same activity list. Thus, associated with the same activity list, we can generate different schedules and different activity set list according to the scheduling mode used.

A similar result to the one derived for the forward scheduling mode holds for the backward mode. If the activity set list being (PRMTb) generated with ConstructASLPRMTb includes a unique subset, the optimal schedule can be obtained with the CPM method.

In [15], Klein introduces the notion of *bidirectional planning* mode where each activity is either scheduled with the forward or with the backward mode. An extension of the activity set list representation may be explored for this mode.

5. Numerical advantages of using Activity Set List Representation

Consider any local search technique where neighbour activity lists of AL are generated by selecting an activity and moving it to different positions between its latest predecessor and its earliest successor in AL in order to satisfy the precedence constraints. Now, referring to the corresponding activity set list generated by the ConstructASLPRMT procedure initialized with AL,

$$ASL = [SA_1, SA_2, \cdots, SA_p]$$

it follows that AL corresponds to a vector

$$V_0 = [os_1, os_2, \cdots, os_p]$$

for some ordering $os_1, os_2, \cdots, os_p \in OS_1 \times OS_2 \times \cdots \times OS_p$. If the selected activity belongs to SA_j, then moving it to any other position within SA_j to maintain the precedence constraints satisfied, corresponds to generate a different $os_j \in OS_j$, but the (PRMT) property induces that this neighbour activity list correspond to the same schedule. It follows that this activity list is not an interesting neighbour to reduce the value of the objective function. Hence the effort can be reduced since the (PRMT) property allows to know beforehand that such an activity list should not be generated and decoded to evaluate its makespan.

To illustrate the reduction in computational effort, consider the first instance of J30 in the PSPLIB library [22] having 30 non dummy activities, and the activity list in Figure 6 generated with the MINSLK (Minimum Job Slack) priority rule [10]. Note that the first and the last activities 1 and 32, are dummy starting and ending activities.

The corresponding activity set list generated with the ConstructASLPRMT procedure is illustrated in Figure 7.

[1, 4, 10, 16, 2, 9, 11, 21, 26, 5, 15, 6, 3, 13, 18, 20, 25, 7, 8, 27, 28, 31, 19, 29, 12, 14, 17, 22, 23, 24, 30, 32]

FIGURE 6. Activity list generated with MINSLK rule considering the first instance of J30.

{1, 4*, 10, 16, 2, **9**, 11, 21, 26}{5, 15}{6}{3, 13}{18, 20, 25, 7, 8}{27, 28, 31, 19}{29, 12, 14*}{17, 22, 23, 24, 30, 32}

FIGURE 7. Activity set list generated with the activity list in Figure 6.

Suppose that the activity 9 is selected. It can be moved to any position after 4 and before 14 to maintain the precedence constraints satisfied ("*" denotes the first and the last positions where activity 9 can be moved). Hence 22 different neighbour activity lists can be generated according to these modifications. But moving activity 9 between activities 4 and 5 (*i.e.*, within its subset SA_1) generates different order lists in OS_1 inducing 6 different activity lists corresponding to the same schedule. Hence, 27% of the 22 neighbour activity lists are useless since they correspond to the same schedule and do not have to be generated and decoded.

Additional tests have been completed with instances of the PSPLIB library as follows : for each instance of the sets J30, J60, J90 (including 30, 60 and 90 non dummy activities, respectively),

1. One hundred different activity lists have been generated randomly according to the RAN priority rule (activities are selected randomly to construct valid activity lists);
2. For each activity list, the experimentation with an activity selected randomly is repeated 100 times;
3. For each selected activity, the percentage of useless neighbour activity lists has been evaluated.

A percentage of useless neighbour activity lists is evaluated for each instance. Referring to [22], recall that each instance of PSPLIB is generated according to parameter settings (network complexity NC, resource factor RF and resource strength RS) that reflect its "complexity". Each set J30, J60 and J90 includes 48 different groups of 10 instances generated with the same parameter settings. We calculate an average percentage of useless neighbour activity lists for each of these groups. The results are summarized in Table 2. The column *Avg. per params* includes the average of the percentage of useless neighbours over the three sets of instances J30, J60 and J90. To simplify the analysis, we have sorted the results in a decreasing order of the values in *Avg. per params*.

Table 2 indicates how the ASL can be useful to reduce the size of the neighborhood. *The percentage of useless neighbours varies between* 14.99% *and* 86.49%. The average value of this percentage is equal to 48.22%, 46.61% and 46.33% for J30, J60 and J90, respectively. Hence, neither the average value nor the percentage of useless neighbours for any parameter settings in Table 2 seem to vary significantly with the size of the problem.

TABLE 2. Percentage of useless neighbours per parameter combination for J30, J60 and J90.

NC	RF	RS	J30 %	J60 %	J90 %	Avg. per params. %
2.10	1.00	1.00	86.49	82.82	80.98	83.43
2.10	0.25	1.00	85.64	82.31	81.03	82.99
2.10	0.50	1.00	85.95	81.74	81.25	82.98
2.10	0.75	1.00	85.30	82.12	81.39	82.94
1.80	1.00	1.00	82.62	79.92	80.54	81.03
1.80	0.75	1.00	81.97	80.74	80.20	80.97
1.80	0.50	1.00	81.41	80.05	80.64	80.70
1.80	0.25	1.00	81.65	79.93	80.13	80.57
1.50	0.50	1.00	78.54	78.36	78.57	78.49
1.50	0.75	1.00	79.68	77.87	77.28	78.28
1.50	1.00	1.00	78.41	77.73	78.33	78.16
1.50	0.25	1.00	77.23	78.49	76.96	77.56
2.10	0.25	0.70	59.71	53.03	54.06	55.60
1.80	0.25	0.70	54.36	52.54	49.11	52.01
2.10	0.50	0.70	53.47	50.28	50.33	51.36
2.10	0.25	0.50	59.15	45.38	45.66	50.06
2.10	0.75	0.70	45.51	49.41	48.88	47.94
2.10	1.00	0.70	46.62	48.29	48.47	47.79
1.80	0.50	0.70	47.14	49.67	46.23	47.68
1.80	1.00	0.70	43.11	47.04	48.44	46.19
1.50	0.25	0.70	45.99	45.55	46.54	46.03
1.80	0.75	0.70	45.82	45.54	46.26	45.87
1.80	0.25	0.50	51.38	42.92	41.83	45.38
1.50	1.00	0.70	40.86	44.35	44.80	43.34
1.50	0.50	0.70	43.22	42.97	42.82	43.01
1.50	0.75	0.70	39.23	42.47	42.98	41.56
2.10	0.50	0.50	46.54	38.29	38.31	41.05
1.50	0.25	0.50	41.80	37.82	36.00	38.54
2.10	0.75	0.50	40.15	36.17	38.41	38.25
1.80	0.50	0.50	40.34	36.26	35.05	37.22
2.10	1.00	0.50	35.98	35.82	37.18	36.33
1.80	1.00	0.50	36.35	36.64	35.91	36.30
1.80	0.75	0.50	36.79	33.97	36.09	35.62
2.10	0.25	0.20	40.68	34.66	30.79	35.37
1.50	0.75	0.50	32.76	32.90	33.41	33.02
1.80	0.25	0.20	40.36	30.85	27.83	33.01
1.50	0.50	0.50	32.55	33.37	32.74	32.88
1.50	1.00	0.50	30.89	33.57	33.58	32.68
1.50	0.25	0.20	31.08	27.30	23.98	27.45
2.10	0.50	0.20	25.72	22.31	22.33	23.45
1.80	0.50	0.20	24.71	21.92	20.66	22.43
1.50	0.50	0.20	20.05	19.42	18.97	19.48
1.80	0.75	0.20	17.58	18.52	19.21	18.43
2.10	0.75	0.20	17.07	18.64	18.05	17.92
2.10	1.00	0.20	16.34	16.96	18.10	17.14
1.80	1.00	0.20	16.03	17.07	18.18	17.09
1.50	0.75	0.20	14.99	17.25	17.51	16.58
1.50	1.00	0.20	15.43	16.28	18.03	16.58

Furthermore, on the one hand the results in Table 2 indicate that the percentage of the useless neighbours does not seem to vary significantly with the complexity factor NC or with the resource factor RF but quite significantly with the resource strength factor RS. On the other hand, the Table 3 allows also to evaluate more clearly the impact of the three parameters. This table includes the average value (columns *Avg. Useless. Mov.*) of the *Avg. per params* values in Table 2 for each value of the parameters NC, RF and RS. The results in Table 3 indicate that *this value increases with the value of NC and RS and decreases with the value of RF.*

TABLE 3. Average percentages of useless neighbours for NC, RF and RS values.

NC	Avg. Useless Mov. %	RF	Avg. Useless. Mov. %	RS	Avg. Useless. Mov.%
1.5	43.98	0.25	52.05	0.2	22.08
1.8	47.53	0.50	46.73	0.5	38.11
2.1	49.66	0.75	44.78	0.7	47.36
		1	44.67	1	80.67

The significant impact if the parameter RS can be explained by the fact that this parameter measures the relationship between the resource demand of the activities and the resource availability [22]. Hence when the resource availabilities increase with regards to the activity demands, then the size of the subsets in a ASL increases and the number of alternate positions where any activity can be moved outside its subset decreases.

Note that the restriction of moving an activity only at positions outside of its subset eliminates many useless movements but not necessarily all of them. Indeed, such a movement may also induce a new activity list corresponding to the same schedule.

6. Conclusion

We present a new Activity Set List representation for (RCPSP). The most efficient heuristics for the problem are based on the activity list representation. We indicate that the Activity Set List can be seen as a generalization of the activity list, and that the same Activity Set List can be associated with several different activity lists corresponding to a same unique schedule. This property allows to specify a new category of heuristics that are searching in a reduced feasible set.

A performance evaluation is completed with PSPLIB, the most used library of instances in the literature. The results indicate clearly how the search space induced by activity lists can be significantly reduced, and hence how interesting solutions can be more quickly reached with procedures using this new representation.

Further investigations are required to explore more completely the impact of this new representation on heuristic procedures.

References

[1] J. Alcaraz and C. Maroto, A Robust Genetic Algorithm for Resource Allocation in Project Scheduling, *Ann. Oper. Res.* **102** (2001) 83–109.

[2] T. Baar, P. Brucker and S. Knust, *Tabu-search algorithms for the resource-constrained project scheduling problem*, Metaheuristics : Advances and Trends in Local Search Paradigms for Optimisation, Kluwer (1997) 1–18.

[3] J. Blazewicz, J.K. Lenstra, A.H.G. and Rinnooy Kan, Scheduling projects to resource constraints : Classification and complexity, *Disc. Appl. Math.* **5** (1983) 11–24.

[4] F.F. Boctor, Some effcient multi-heuristic procedures for resource-constrained project scheduling, *Eur. J. Oper. Res.* **49** (1990) 3–13.
[5] F.F. Boctor, Resource-constrained project scheduling by simulated annealing, *Int. J. Prod. Res.* **34** (1996) 2335–2351.
[6] M. Bouleimen, H. Lecocq, A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple version, *Eur. J. Oper. Res.* **149** (2003) 268–281.
[7] P. Brucker, A. Drexl, R. Mohring, K. Neumann, Resource-constrained project scheduling : Notation, classification, models and methods, *Eur. J. Oper. Res.* **112** (1999) 3–41.
[8] J.-H. Cho, Y.-D. Kim, A simulated annealing algorithm for resource-constrained project scheduling problems, *J. Oper. Res. Soc.* **48** (1997) 735–744.
[9] E.W. Davis, G.E. Heidorn, An algorithm for optimal project scheduling under multiple resource constraints, *Manage. Sci.* **27** (1971) B803–B816.
[10] E.W. Davis, J.H. Patterson, A comparison of heuristic and optimal solutions in resource-constrained project scheduling, *Manage. Sci.* **21** (1975) 944–955.
[11] E. Demeulemeester, W. Harroelen, New benchmarking results for the resource constrained project scheduling problem, *Manage. Sci.* **43** (1995) 1485–1492.
[12] E. Demeulemeester, W. Herroelen, A branch-and-bound procedure for multiple resource-constrained project scheduling problem, *Manage. Sci.* **38** (1992) 1803–1818.
[13] S. Hartmann, A competitive genetic algorithm for resource-constrained project scheduling, *Nav. Res. Logist.* **456** (1998) 733–750.
[14] W. Herroelen, B. Reyck, E. Demeulemeester, Resource-constrained project scheduling : A survey of recent developments, *Computers and Operations Research* **4** (1998) 279–302.
[15] R. Klein, Bidirectional planning : improving priority rule-based heuristics for scheduling resource-constrained projects, *Eur. J. Oper. Res.* **127** (2000) 619–638.
[16] U. Kohlmorgen, H. Schmeck, K. Haase, Experiences with fine-grained parallel genetic algorithms, *Ann. Oper. Res.* **90** (1999) 203–219.
[17] R. Kolisch, A. Drexel, Adaptative search for solving hard project scheduling problem, *Nav. Res. Logist.* **43** (1996) 23–40.
[18] R. Kolisch, Serial and Parallel resource-constrained project scheduling methods revisited: Theory and computation, *Eur. J. Oper. Res.* **90** (1996) 320–333.
[19] R. Kolisch, Efficient priority rules for the resource-constrained project scheduling problem, *J. Oper. Manage.* **14** (1996) 179–192.
[20] R. Kolisch, A. Drexl, Adaptative search for solving hard project scheduling problems, *Nav. Res. Logist.* **43** (1996) 23–40.
[21] R. Kolisch, S. Hartmann, Heuristic algorithms for solving resource-constrained project scheduling problem : Classification and computation analysis, in : *Project Scheduling : Recent Models, Algorithms and Applications*, edited by J. Weglarz, Kluwer Academic Publisher, Boston 1999, 147–178.
[22] Kolisch R., Sprecher A., Drexel A., Characterization and generation of general class of resource-constrained project scheduling problems, *Manage. Sci.* **41** (1995) 1693–1703.
[23] A. Mingozi, V. Maniezzo, S. Ricciardelli, L. Bianco, An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation, *Manage. Sci.* **44** (1998) 714–729.
[24] J.H. Patterson, R. Sowinski, F.B. Talbot, J. Weglarz, An algorithm for a general class of precedence and resource constrained scheduling problems, in : *Advances in project scheduling*, edited by R. Sowinski, J. Weglarz, Elsevier, Amsterdam 1989, 3–28.
[25] E. Pinson, C. Prins, F. Rullier, Using tabu search for solving the resource-constrained project scheduling problem, in: *Proceedings of the 4th International Workshop on Project Management and Scheduling*, Leuven, Belgium 1994, 102–106.
[26] A. Sprecher, R. Kolisch, A. Drexel, Semi-Active, active and non-delay schedules for resource-constrained project scheduling problem, *Eur. J. Oper. Res.* **80** (1995) 94–102.
[27] J.P. Stinson, E.W. Davis, B.M. Khumawala, Multiple resource-constrained scheduling using branch-and-bound, *AIIE Transactions* **10** (1978) 252–259.

[28] B. Talbot, J.H. Patterson, An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problem, *Manage. Sci.* **24** (1978) 1163–1174.
[29] A. Thesen, Heuristic scheduling of activities under resource and precedence restrictions, *Manage. Sci.* **23** (1976) 412–422.